Lab 6

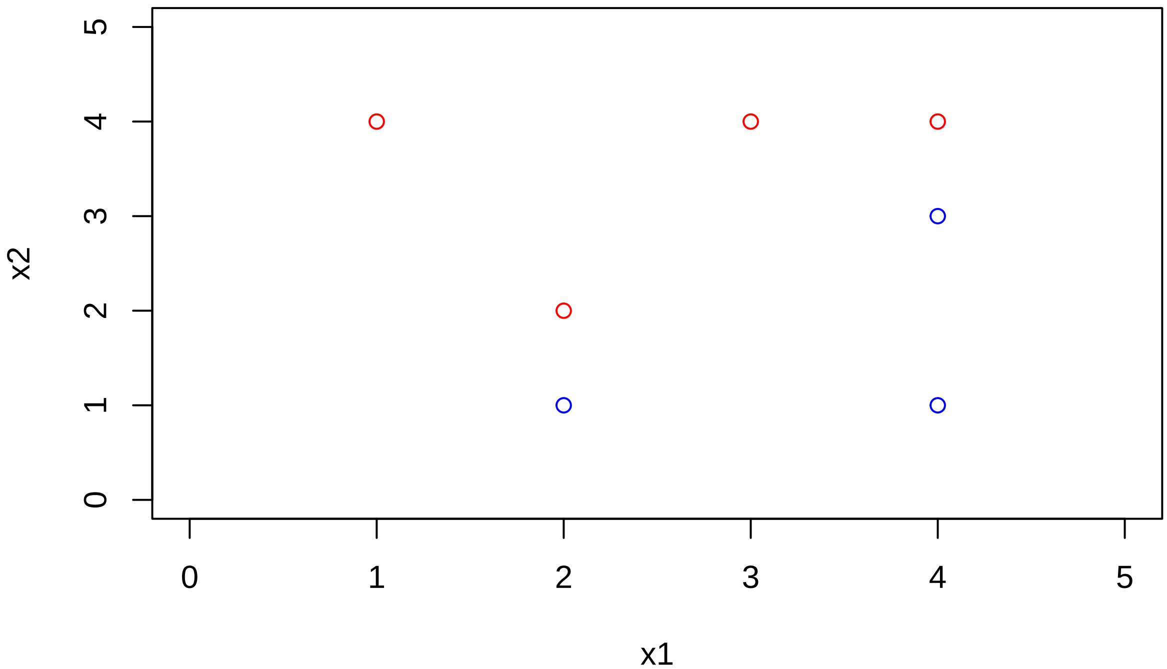
Zara Waheed

March 12, 2022

Question 9.3

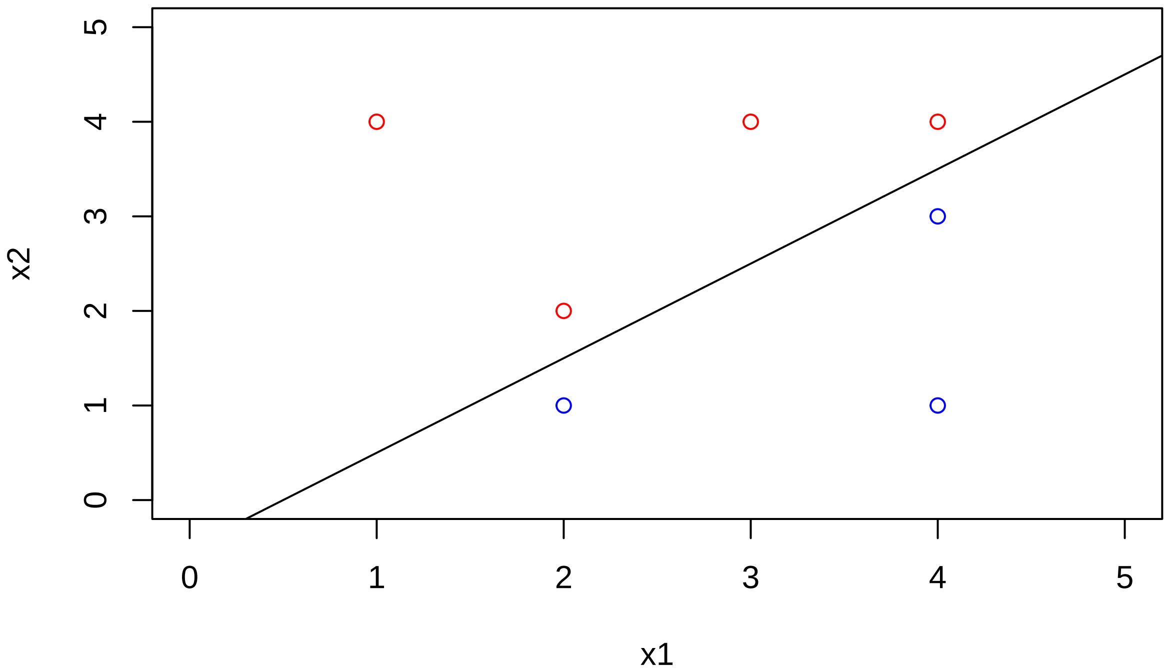
a)

```
x1 = c(3,2,4,1,2,4,4)
x2 = c(4,2,4,4,1,3,1)
colors = c("red", "red", "red", "red", "blue", "blue", "blue")

plot(x1,x2,col=colors,xlim=c(0,5),ylim=c(0,5))
```



b)

```
plot(x1, x2, col=colors, xlim=c(0,5), ylim=c(0,5))
abline(-0.5, 1)
```

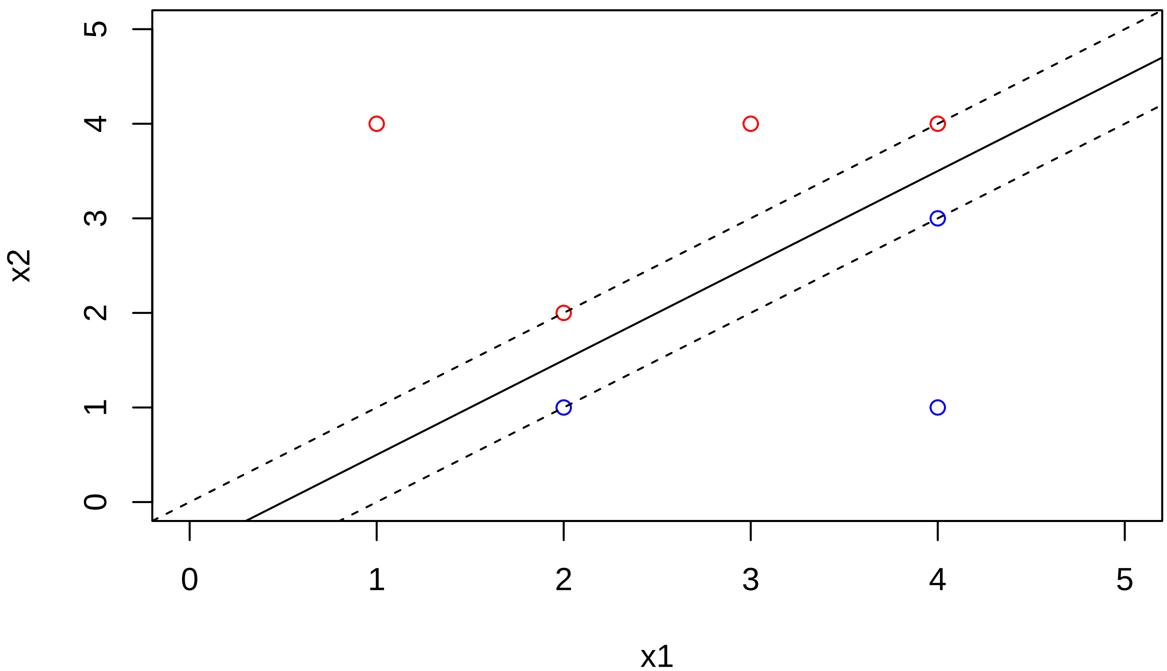


c)

Classify as red if $0.5 - X1 + X2 > 0$ Classify as blue if $0.5 - X1 + X2 < 0$

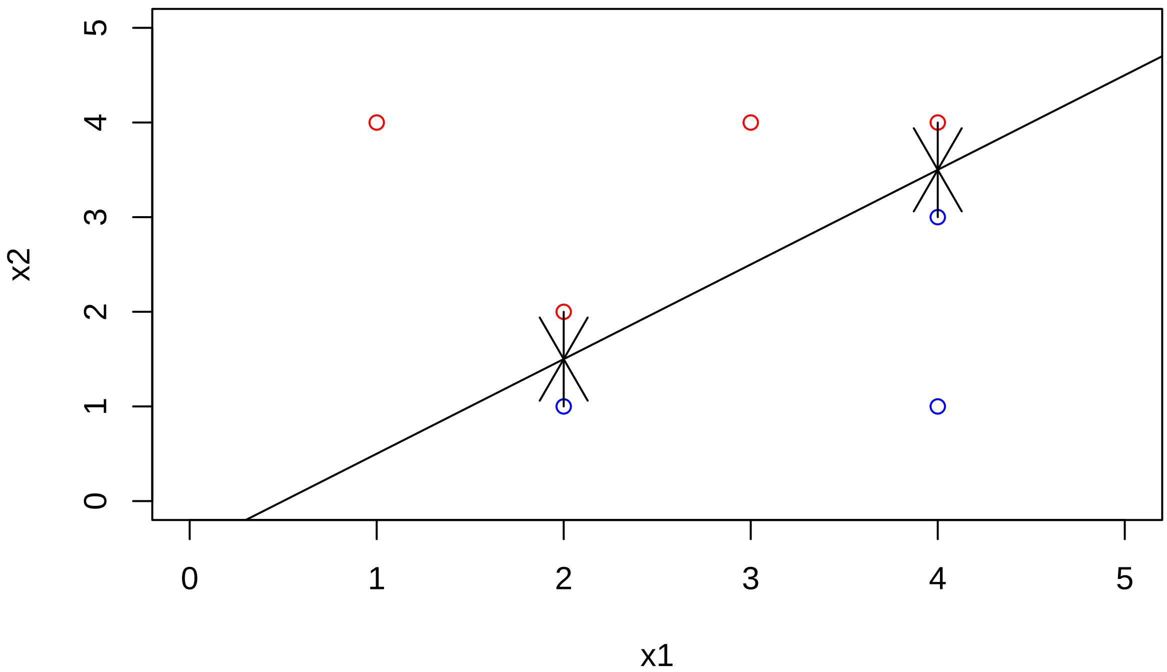
d)

```
plot(x1,x2,col=colors,xlim=c(0,5),ylim=c(0,5))
abline(-0.5, 1)
abline(-1, 1, lty=2)
abline(0, 1, lty=2)
```



e)

```
plot(x1, x2, col=colors, xlim=c(0,5), ylim=c(0,5))

abline(-0.5, 1)
arrows(2,1,2,1.5)
arrows(2,2,2,1.5)
arrows(4,4,4,3.5)
arrows(4,3,4,3.5)
```

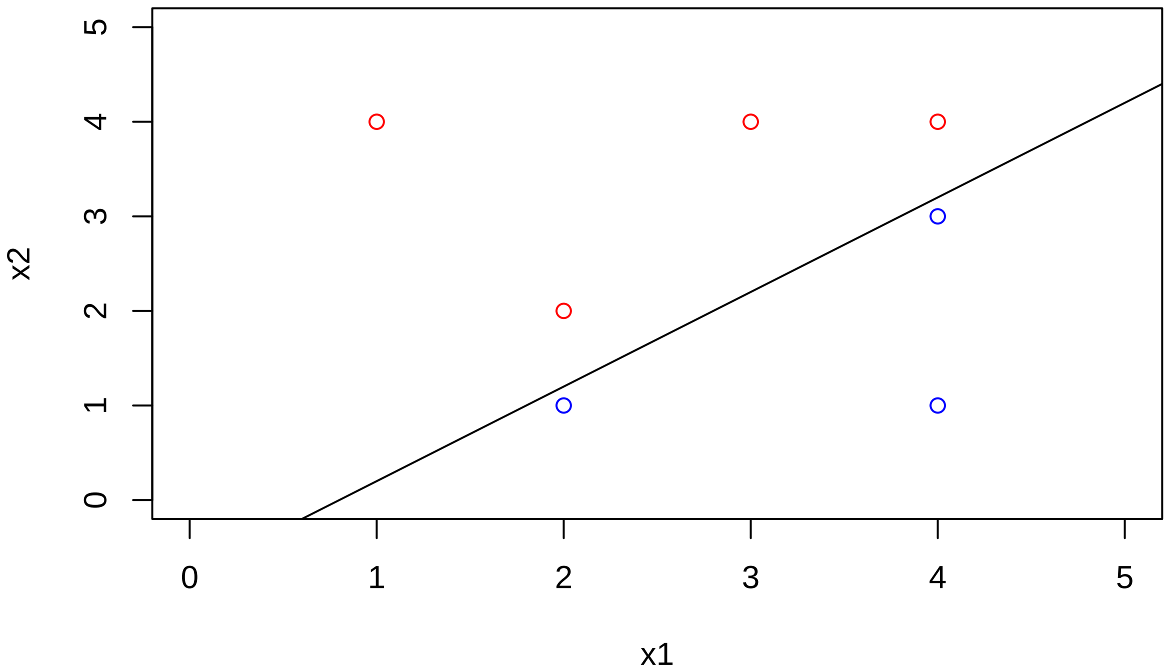


f)

It would not have an effect on the maximal margin hyperplane since its not a support vector.

g)

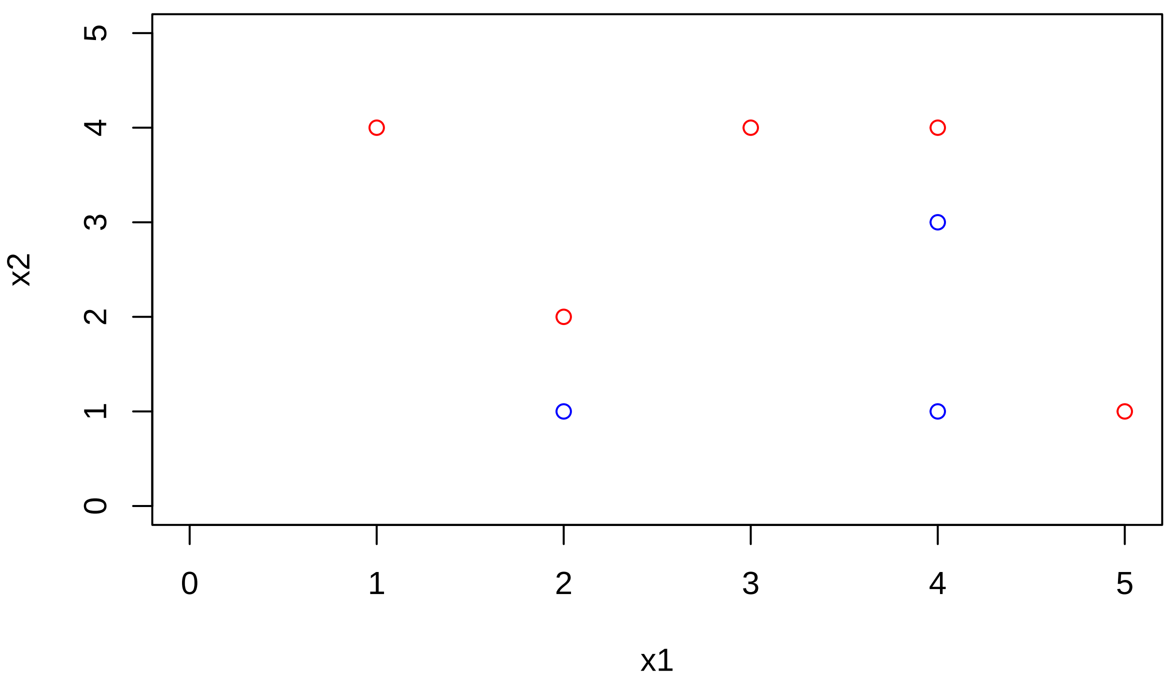
```
plot(x1, x2, col=colors, xlim=c(0,5), ylim=c(0,5))
abline(-0.8, 1)
```



X1 + X2 > 0

h)

```
plot(x1, x2, col=colors, xlim=c(0,5), ylim=c(0,5))
points(c(5), c(1), col=c("red"))
```



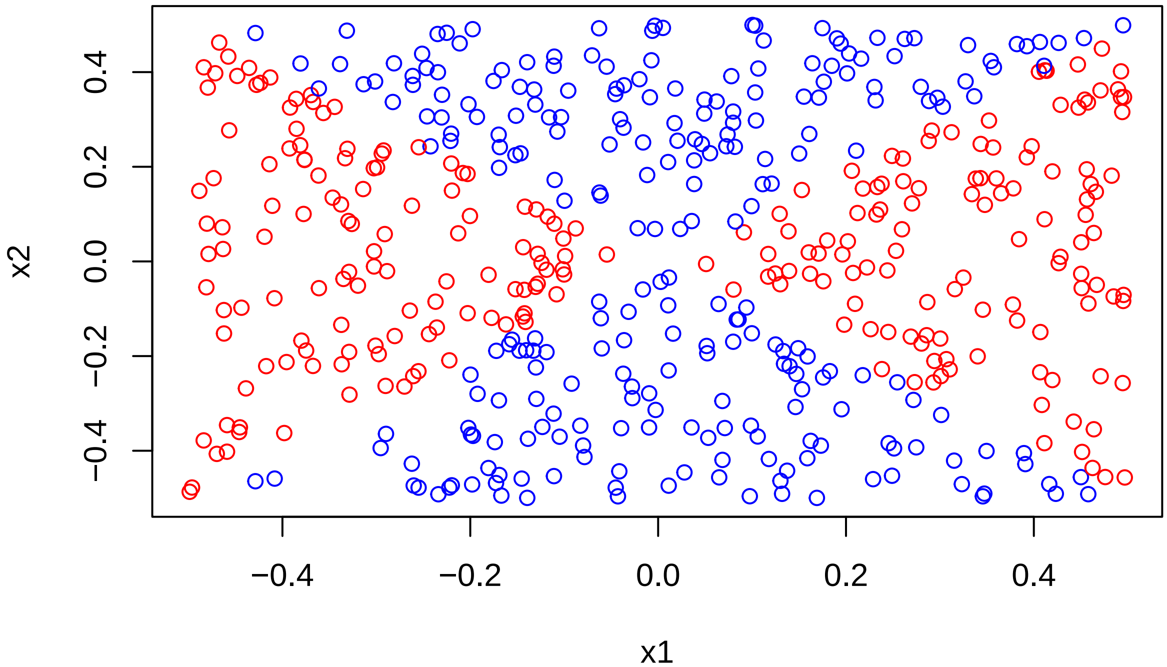
Question 9.5

a)

```
set.seed(100)
x1 <- runif(500) - 0.5
x2 <- runif(500) - 0.5
y <- 1*(x1^2 - x2^2 > 0)
```

b)

```
plot(x1, x2, col = ifelse(y, "red", "blue"))
```

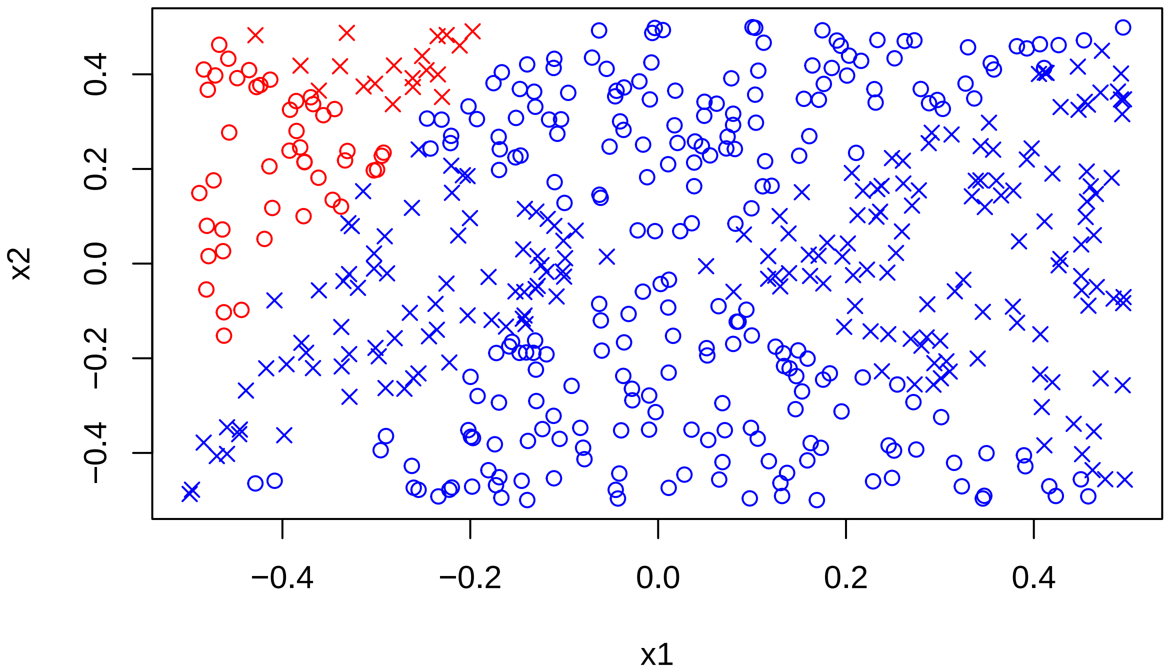


c)

```
df <- data.frame(x1, x2, y)
fit <- glm(y ~ x1 + x2, data = df, family = binomial)
```

d)

```
pred_fit <- predict(fit, data.frame(x1,x2))

plot(x1, x2, col = ifelse(pred_fit > 0, "red", "blue"), pch = ifelse(as.integer(pred_fit > 0) == y, 1,4
```



circles: correctly classified crosses: incorrectly classified

e)

```
fit1 <- glm(y ~ poly(x1, 2) + poly(x2, 2), data = df, family = binomial)
summary(fit1)
```

```
##
## Call:
## glm(formula = y ~ poly(x1, 2) + poly(x2, 2), family = binomial,
##     data = df)
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -1.535e-03  -2.000e-08  -2.000e-08   2.000e-08   1.491e-03
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -217.3     6065.2  -0.036    0.971
## poly(x1, 2)1    3871.4   125969.8   0.031    0.975
## poly(x1, 2)2   33920.7   929001.2   0.037    0.971
## poly(x2, 2)1    -606.6    64636.1  -0.009    0.993
## poly(x2, 2)2  -35221.1   965075.4  -0.036    0.971
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6.9276e+02  on 499  degrees of freedom
## Residual deviance: 5.0915e-06  on 495  degrees of freedom
## AIC: 10
##
## Number of Fisher Scoring iterations: 25
```

```r
fit2 <- glm(y ~ x1 + x2 + x1*x2, data = df, family = binomial)
summary(fit2)
```

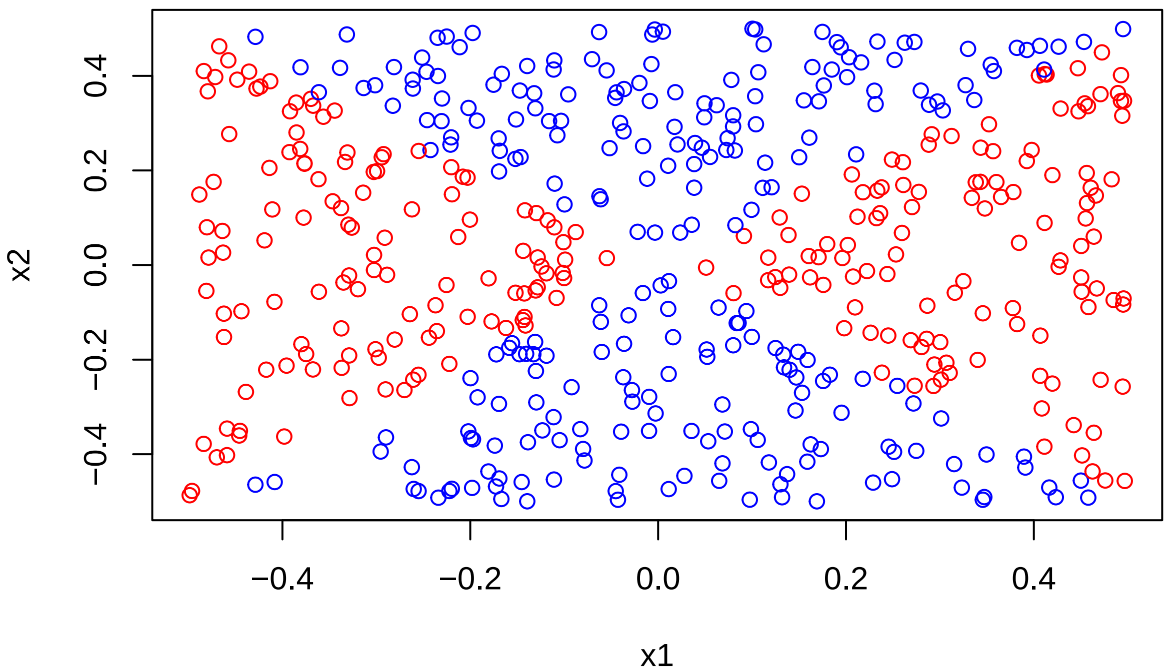
```
##
## Call:
## glm(formula = y ~ x1 + x2 + x1 * x2, family = binomial, data = df)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.202  -1.161  -1.093   1.193   1.283
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.05600    0.08990  -0.623    0.533
## x1          -0.16065    0.32387  -0.496    0.620
## x2           0.05885    0.30404   0.194    0.847
## x1:x2        0.36419    1.06438   0.342    0.732
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 692.76  on 499  degrees of freedom
## Residual deviance: 692.39  on 496  degrees of freedom
## AIC: 700.39
##
## Number of Fisher Scoring iterations: 3
```

```r
fit3 <- glm(y ~ x1 + x2 + log(x1) + log(x2), data = df, family = binomial)
summary(fit3)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2 + log(x1) + log(x2), family = binomial,
##     data = df)
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -3.154e-04  -2.000e-08  -2.000e-08   2.000e-08   3.280e-04
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)    689.959 360999.814   0.002    0.998
## x1            3773.752 874746.558   0.004    0.997
## x2           -4938.869 582522.494  -0.008    0.993
## log(x1)         -8.037 129284.294   0.000    1.000
## log(x2)        253.071  34668.704   0.007    0.994
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1.8573e+02  on 133  degrees of freedom
## Residual deviance: 2.1127e-07  on 129  degrees of freedom
##   (366 observations deleted due to missingness)
## AIC: 10
##
## Number of Fisher Scoring iterations: 25
```

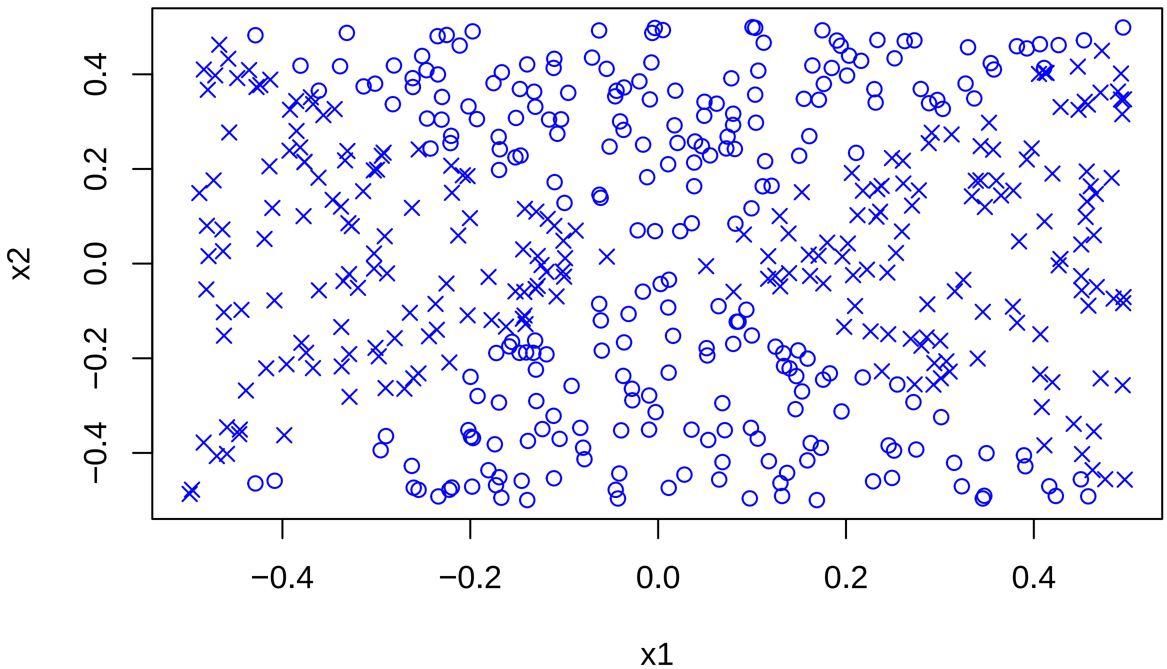
f)

```r
pred_fit <- predict(fit1, df)
plot(x1, x2, col = ifelse(pred_fit > 0, "red", "blue"), pch = ifelse(as.integer(pred_fit > 0) == y, 1,4
```



g)

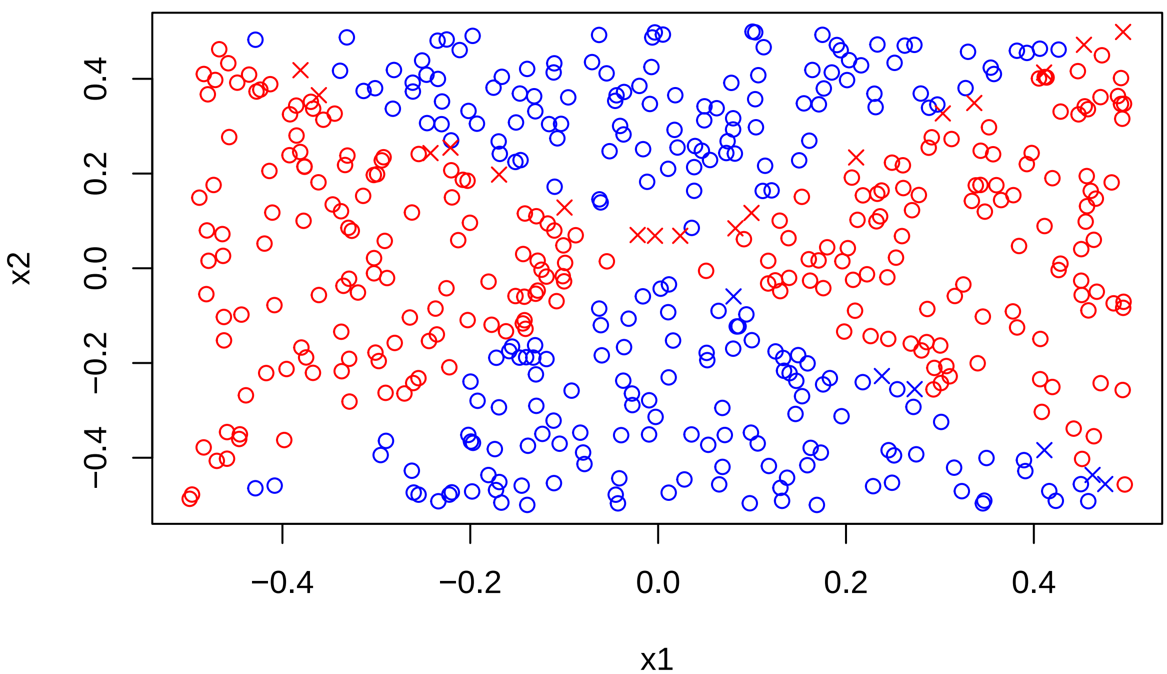
```r
df$y <- as.factor(df$y)
fit_svc <- svm(y ~ x1 + x2, data = df, kernel = "linear")
pred_svc <- predict(fit_svc, df, type = "response")
plot(x1, x2, col = ifelse(pred_svc != 0, "red", "blue"), pch = ifelse(pred_svc == y, 1,4))
```



circles: correctly classified crosses: incorrectly classified

h)

```
fit_svm <- svm(y ~ x1 + x2, data = df, kernel = "polynomial", degree = 2)
pred_svm <- predict(fit_svm, df, type = "response")
plot(x1, x2, col = ifelse(pred_svm != 0, "red", "blue"), pch = ifelse(pred_svm == y, 1,4))
```



i)

Support vector model works better with a non-linear kernel and logistic regression with non-linear predictors.

Question 9.7

##a)

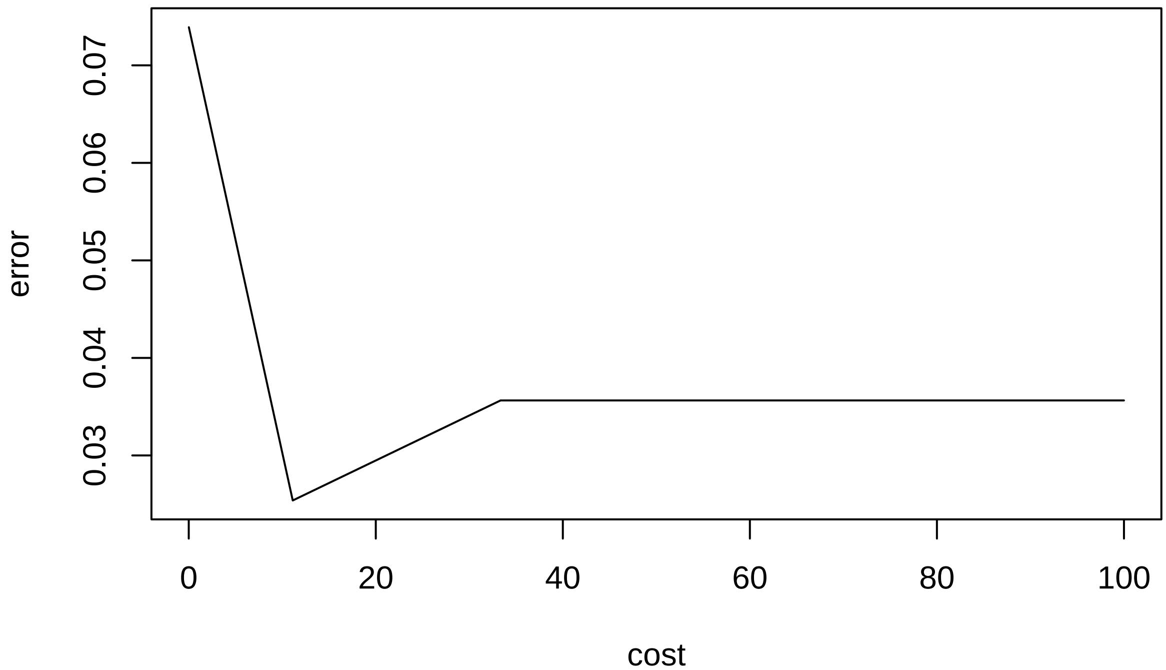
```
data("Auto")
Auto$Y <- ifelse(Auto$mpg > median(Auto$mpg), 1, 0)
Auto$Y <- as.factor(Auto$Y)
```

  (b)

```
set.seed(100)
costrange <- data.frame(cost = seq(0.01, 100, length.out = 10))
svm_auto <- tune(svm, Y ~ ., data = Auto, kernel = "linear", ranges = costrange)

summary(svm_auto)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost
##  11.12
##
## - best performance: 0.02538462
##
## - Detailed performance results:
##      cost      error dispersion
## 1    0.01 0.07391026 0.04398186
## 2   11.12 0.02538462 0.02372507
## 3   22.23 0.03051282 0.02316421
## 4   33.34 0.03564103 0.02125655
## 5   44.45 0.03564103 0.02125655
## 6   55.56 0.03564103 0.02125655
## 7   66.67 0.03564103 0.02125655
## 8   77.78 0.03564103 0.02125655
## 9   88.89 0.03564103 0.02125655
## 10 100.00 0.03564103 0.02125655
```

```
plot(svm_auto$performances[,c(1,2)], type = "l")
```



Keeping the cost at 11.12 seems to perform the best.

(c)

```
# Polynomial
costrange <- data.frame(cost = seq(0.01, 100, length.out = 5), degree = seq(1, 100, length.out = 5))
svm_poly_auto <- tune(svm, Y ~ ., data = Auto, kernel = "polynomial", ranges = costrange)
summary(svm_poly_auto)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##    cost degree
##  50.005      1
##
## - best performance: 0.02544872
##
## - Detailed performance results:
##        cost degree      error dispersion
## 1    0.0100   1.00 0.53410256 0.14822525
## 2   25.0075   1.00 0.04846154 0.03286461
## 3   50.0050   1.00 0.02544872 0.02402692
## 4   75.0025   1.00 0.02544872 0.02076512
## 5  100.0000   1.00 0.02544872 0.02076512
## 6    0.0100  25.75 0.52660256 0.17132309
## 7   25.0075  25.75 0.52660256 0.17132309
## 8   50.0050  25.75 0.52660256 0.17132309
## 9   75.0025  25.75 0.52660256 0.17132309
## 10 100.0000  25.75 0.52660256 0.17132309
## 11   0.0100  50.50 0.58410256 0.03892151
## 12  25.0075  50.50 0.58410256 0.03892151
```

```
## 13  50.0050  50.50 0.58410256 0.03892151
## 14  75.0025  50.50 0.58410256 0.03892151
## 15 100.0000  50.50 0.58410256 0.03892151
## 16   0.0100  75.25 0.52660256 0.17132309
## 17  25.0075  75.25 0.52660256 0.17132309
## 18  50.0050  75.25 0.52660256 0.17132309
## 19  75.0025  75.25 0.52660256 0.17132309
## 20 100.0000  75.25 0.52660256 0.17132309
## 21   0.0100 100.00 0.58410256 0.03892151
## 22  25.0075 100.00 0.58410256 0.03892151
## 23  50.0050 100.00 0.58410256 0.03892151
## 24  75.0025 100.00 0.58410256 0.03892151
## 25 100.0000 100.00 0.58410256 0.03892151
```

Keeping the cost at 75 or 100 with degree 1 seems to perform the best.

```r
# Radial
costrange <- data.frame(cost=seq(0.01,100,length.out = 5),gamma=seq(0.1,100,length.out = 5))
svm_rad_auto <- tune(svm, Y ~ ., data = Auto, kernel = "radial", ranges = costrange)
summary(svm_rad_auto)
```

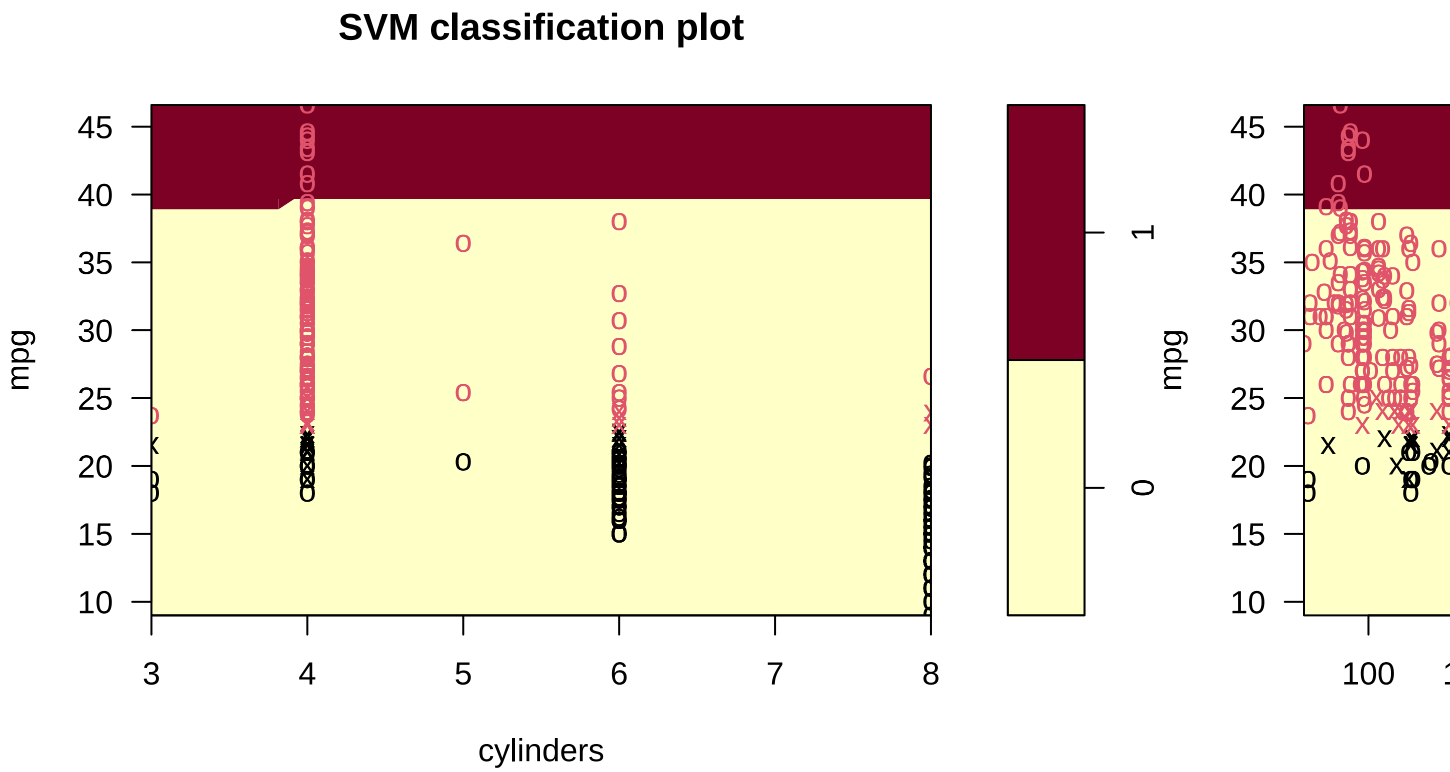
```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##     cost gamma
##  25.0075   0.1
##
## - best performance: 0.02544872
##
## - Detailed performance results:
##        cost   gamma      error dispersion
## 1    0.0100   0.100 0.15564103 0.08291358
## 2   25.0075   0.100 0.02544872 0.02076512
## 3   50.0050   0.100 0.03051282 0.02331940
## 4   75.0025   0.100 0.02794872 0.02216297
## 5  100.0000   0.100 0.02794872 0.02216297
## 6    0.0100  25.075 0.54089744 0.03239108
## 7   25.0075  25.075 0.52557692 0.03725610
## 8   50.0050  25.075 0.52557692 0.03725610
## 9   75.0025  25.075 0.52557692 0.03725610
## 10 100.0000  25.075 0.52557692 0.03725610
## 11   0.0100  50.050 0.54089744 0.03239108
## 12  25.0075  50.050 0.53576923 0.03854905
## 13  50.0050  50.050 0.53576923 0.03854905
## 14  75.0025  50.050 0.53576923 0.03854905
## 15 100.0000  50.050 0.53576923 0.03854905
## 16   0.0100  75.025 0.54089744 0.03239108
## 17  25.0075  75.025 0.53833333 0.03570589
## 18  50.0050  75.025 0.53833333 0.03570589
## 19  75.0025  75.025 0.53833333 0.03570589
## 20 100.0000  75.025 0.53833333 0.03570589
## 21   0.0100 100.000 0.54089744 0.03239108
```

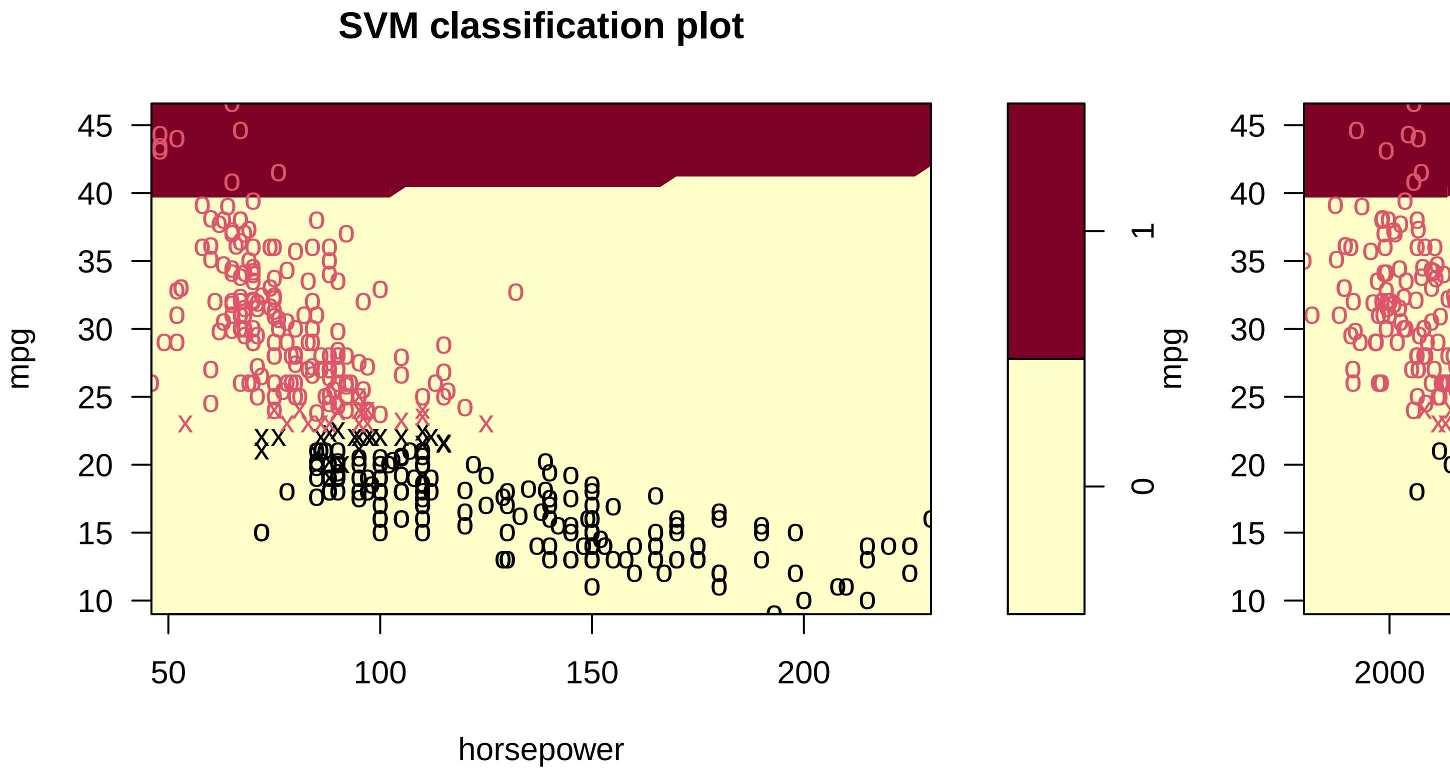
```
## 22  25.0075 100.000 0.54089744 0.03239108
## 23  50.0050 100.000 0.54089744 0.03239108
## 24  75.0025 100.000 0.54089744 0.03239108
## 25 100.0000 100.000 0.54089744 0.03239108
```

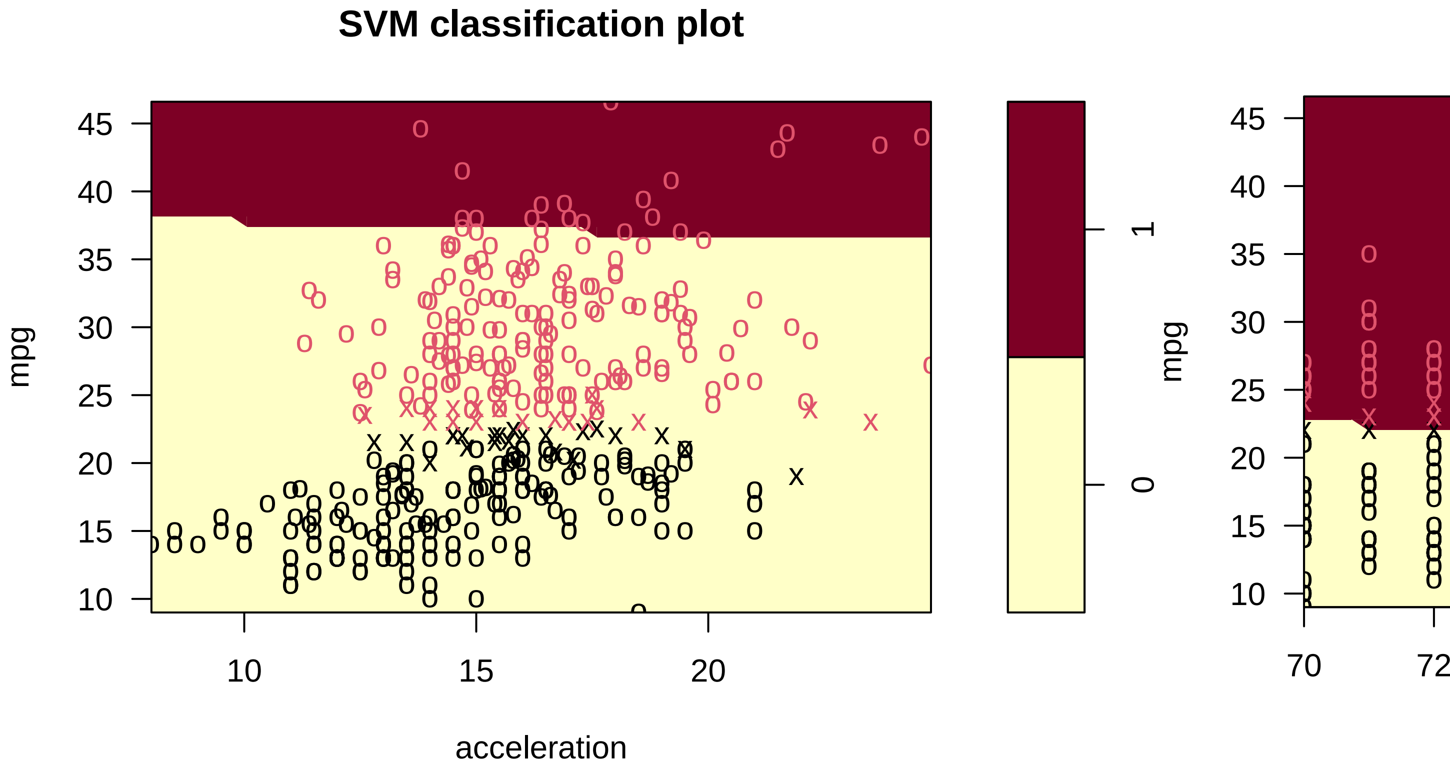
Keeping a cost of 25 with gamma 0.1 seems to perform the best.

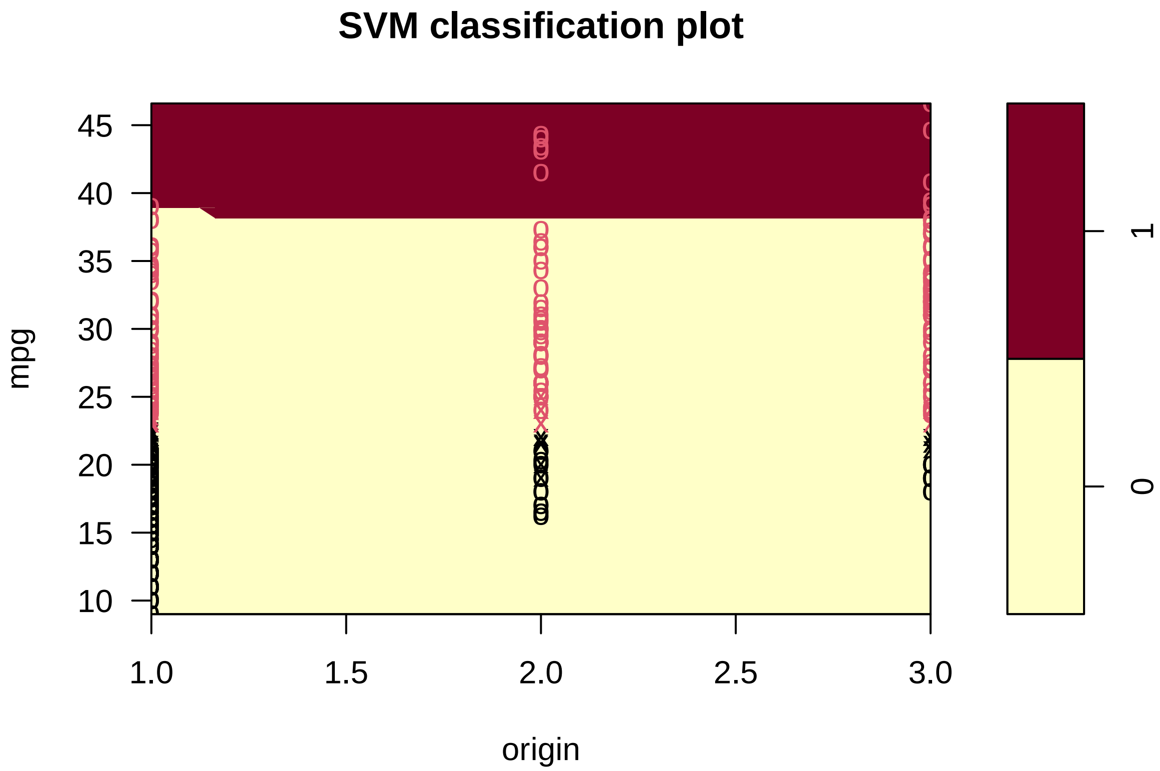
d)

```r
fit1 <- svm(Y ~ ., data = Auto, kernel = "linear", cost = 11.12)
fit2 <- svm(Y ~ ., data = Auto, kernel = "polynomial", cost = 100, degree = 1)
fit3 <- svm(Y ~ ., data = Auto, kernel = "radial", cost = 25, gamma = 0.1)

svm_plot <- function(a){
  for (name in names(Auto)[!(names(Auto) %in% c("mpg", "Y", "name"))])
    plot(a, Auto, as.formula(paste("mpg~", name, sep = "")))
}

# got help from a classmate to create this function

svm_plot(fit1)
```

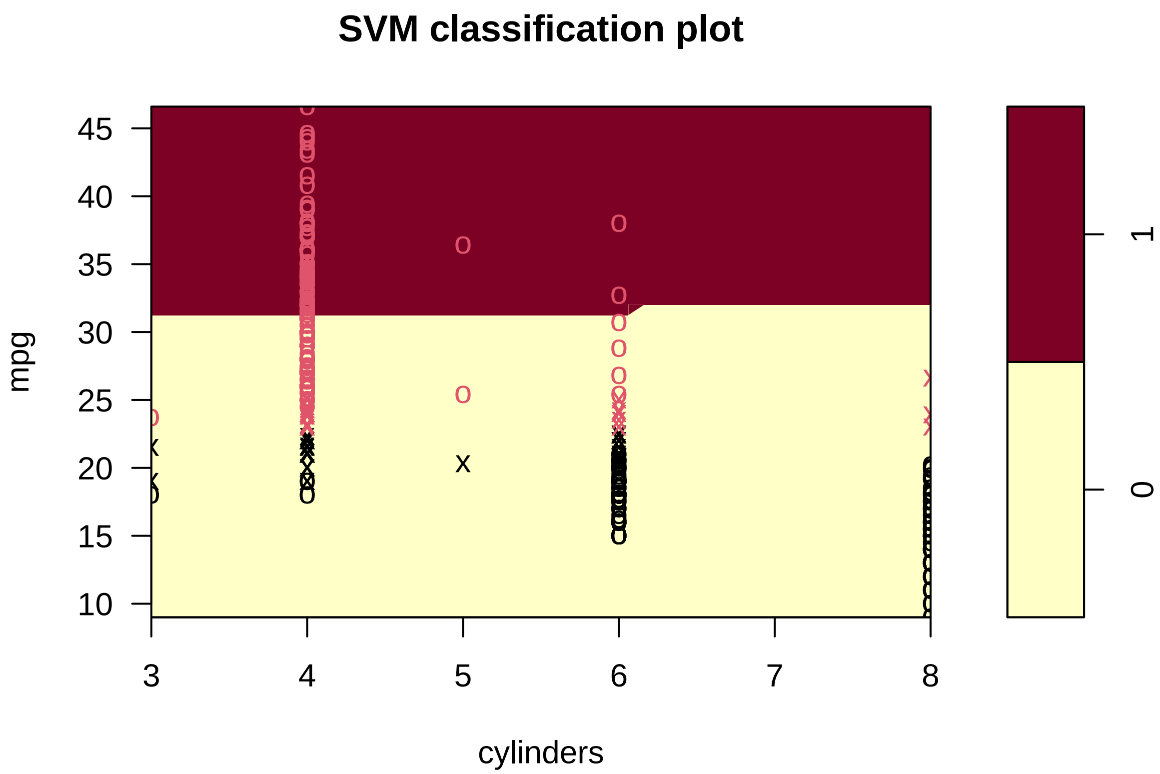


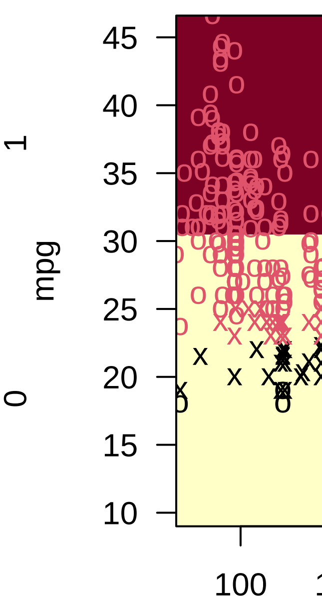

SVM classification plot
mpg
45
40
35
30
25
20
15
10
50
100
150
200
horsepower
1
0
mpg
2000

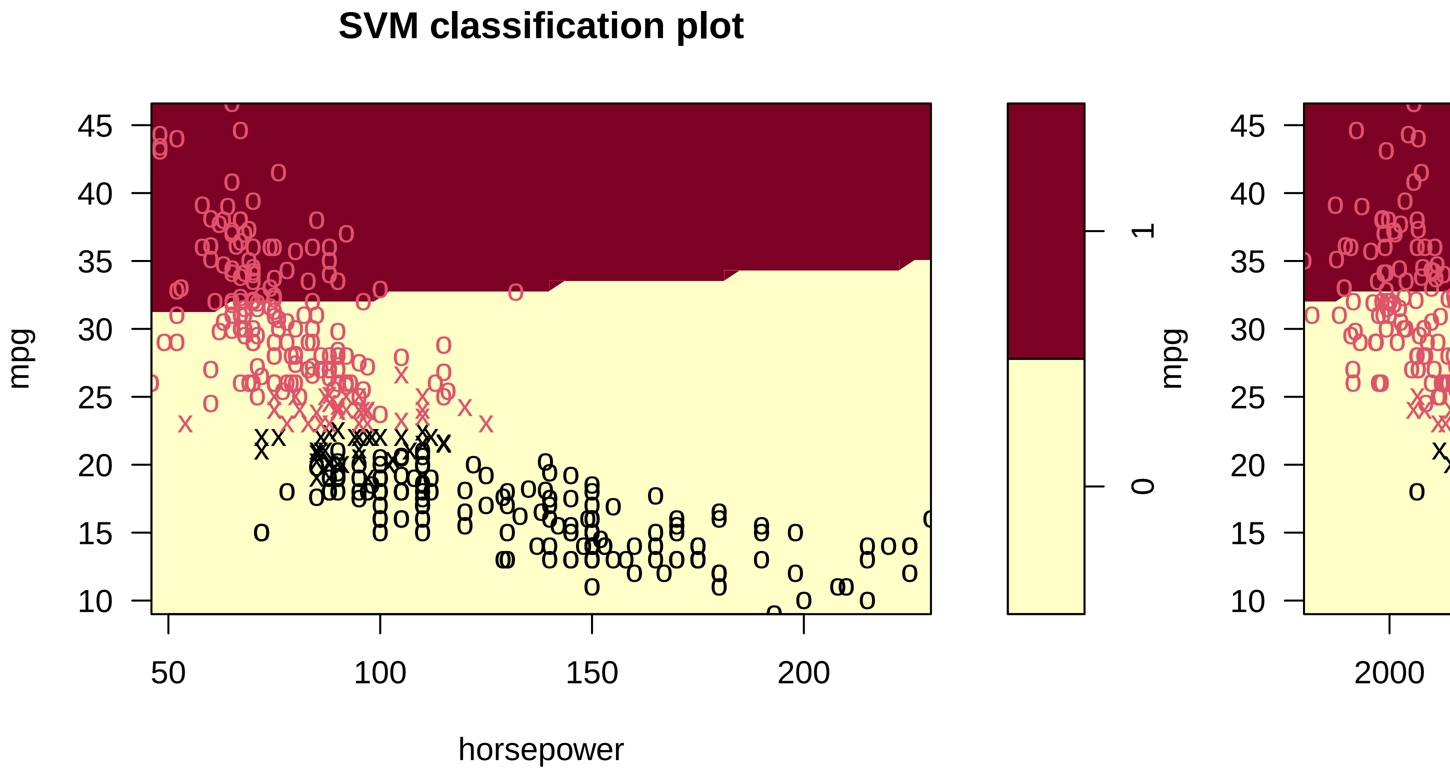

SVM classification plot
mpg
45
40
35
30
25
20
15
10
10
15
20
acceleration
1
0
mpg
70
72

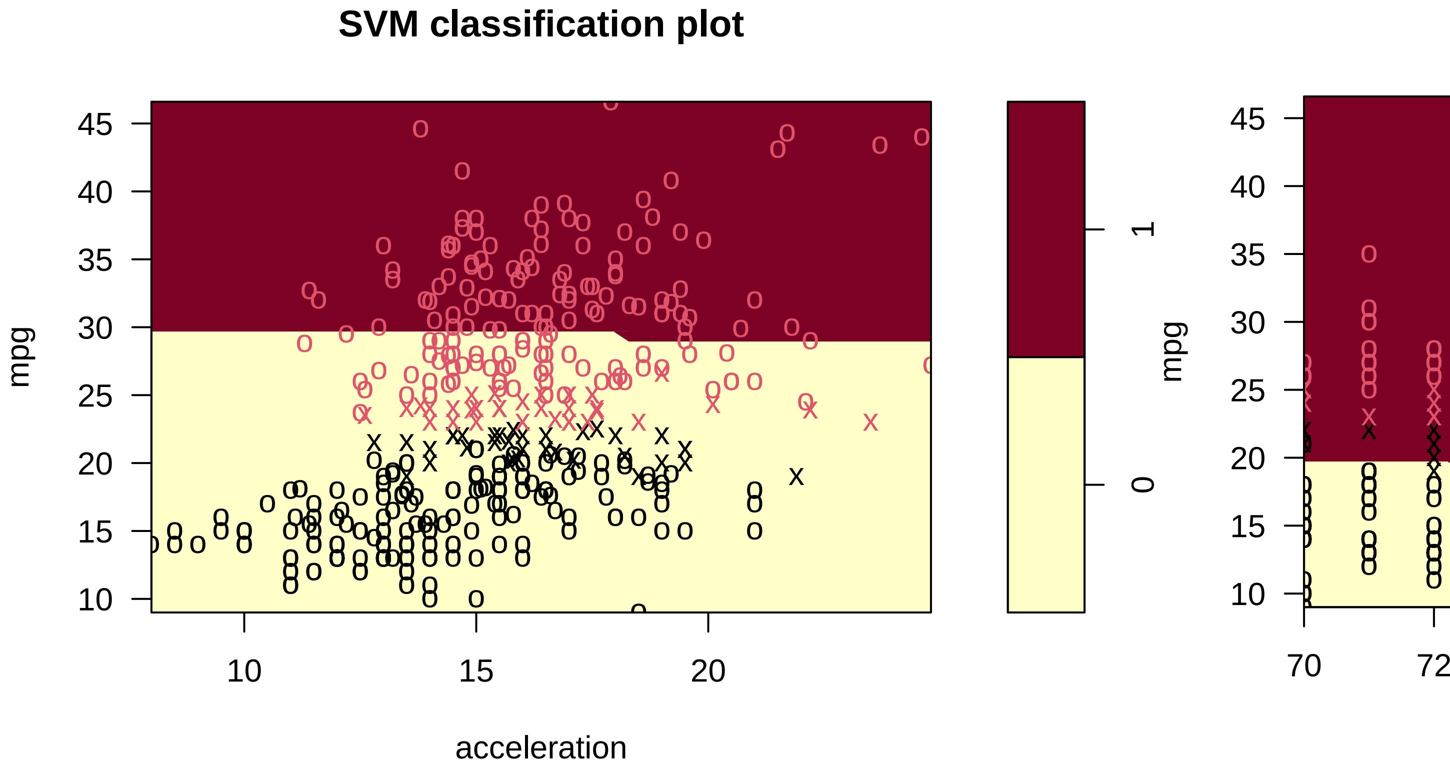


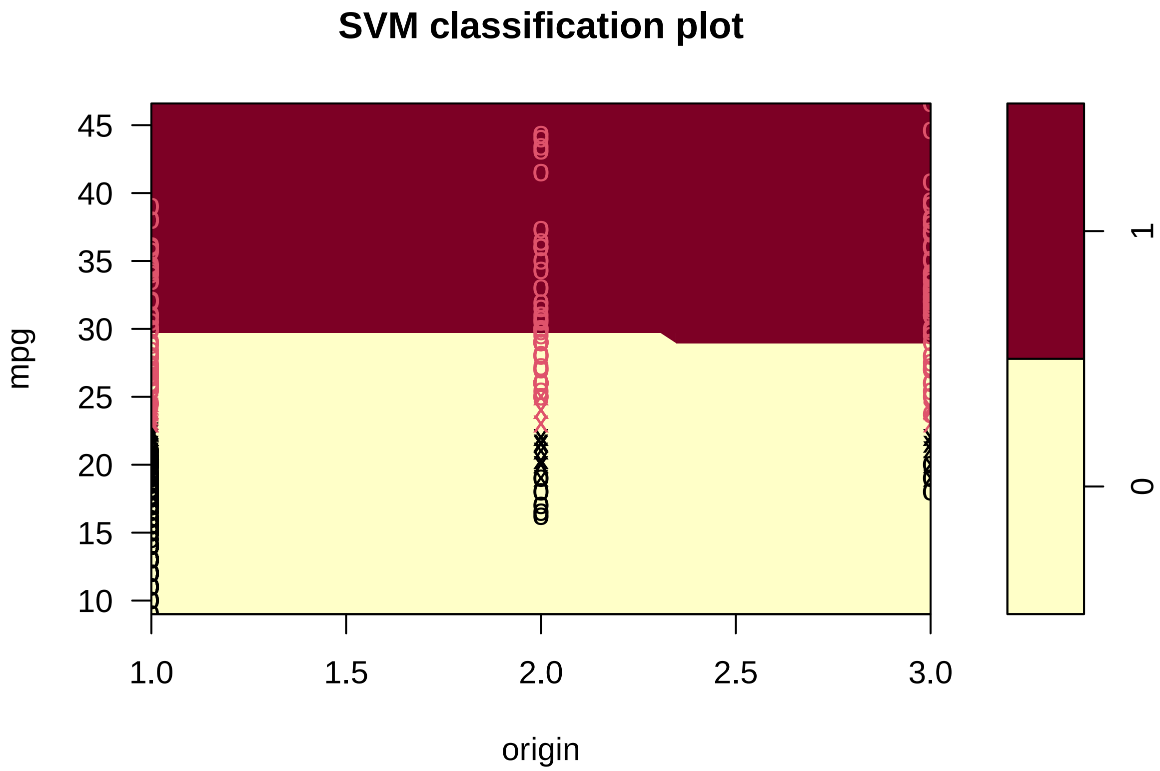

```
svm_plot(fit2)
```



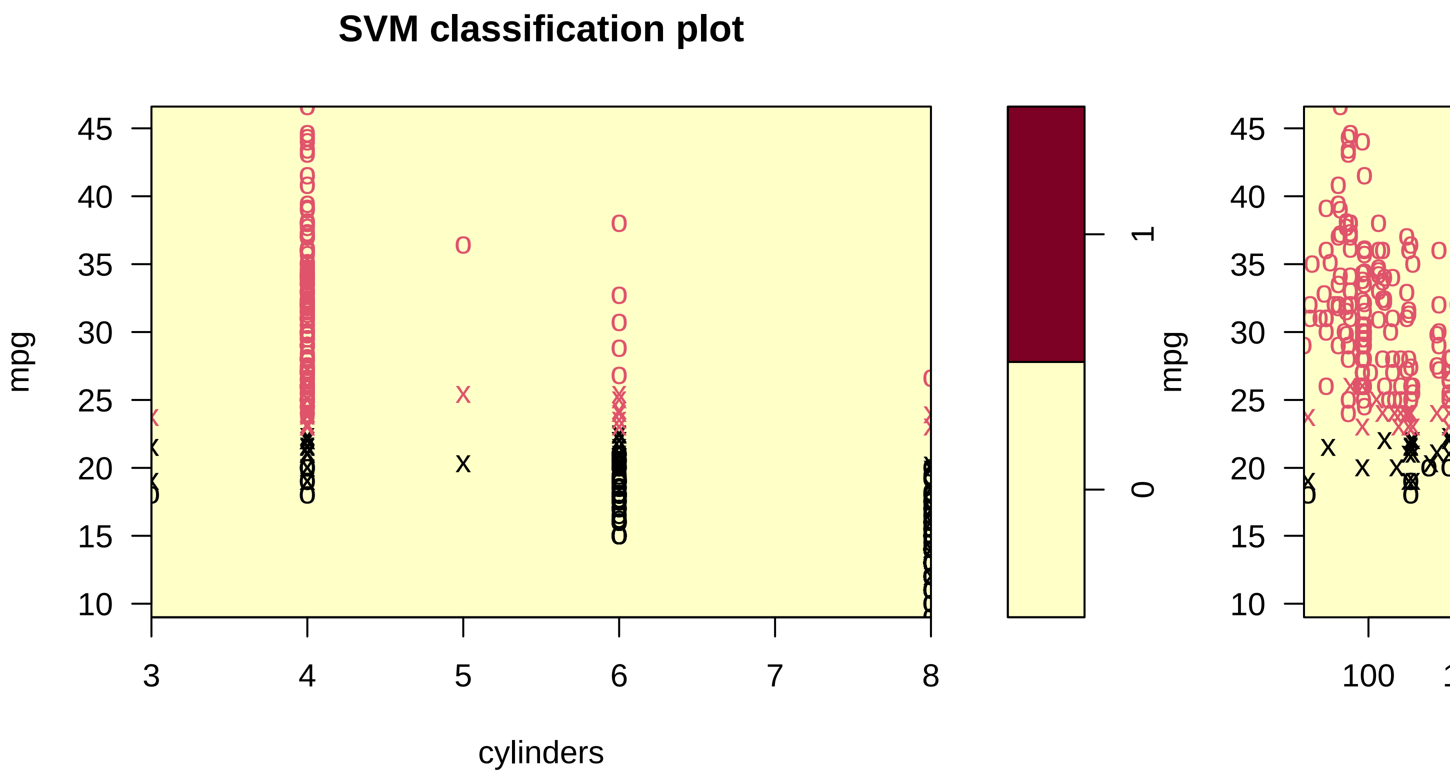


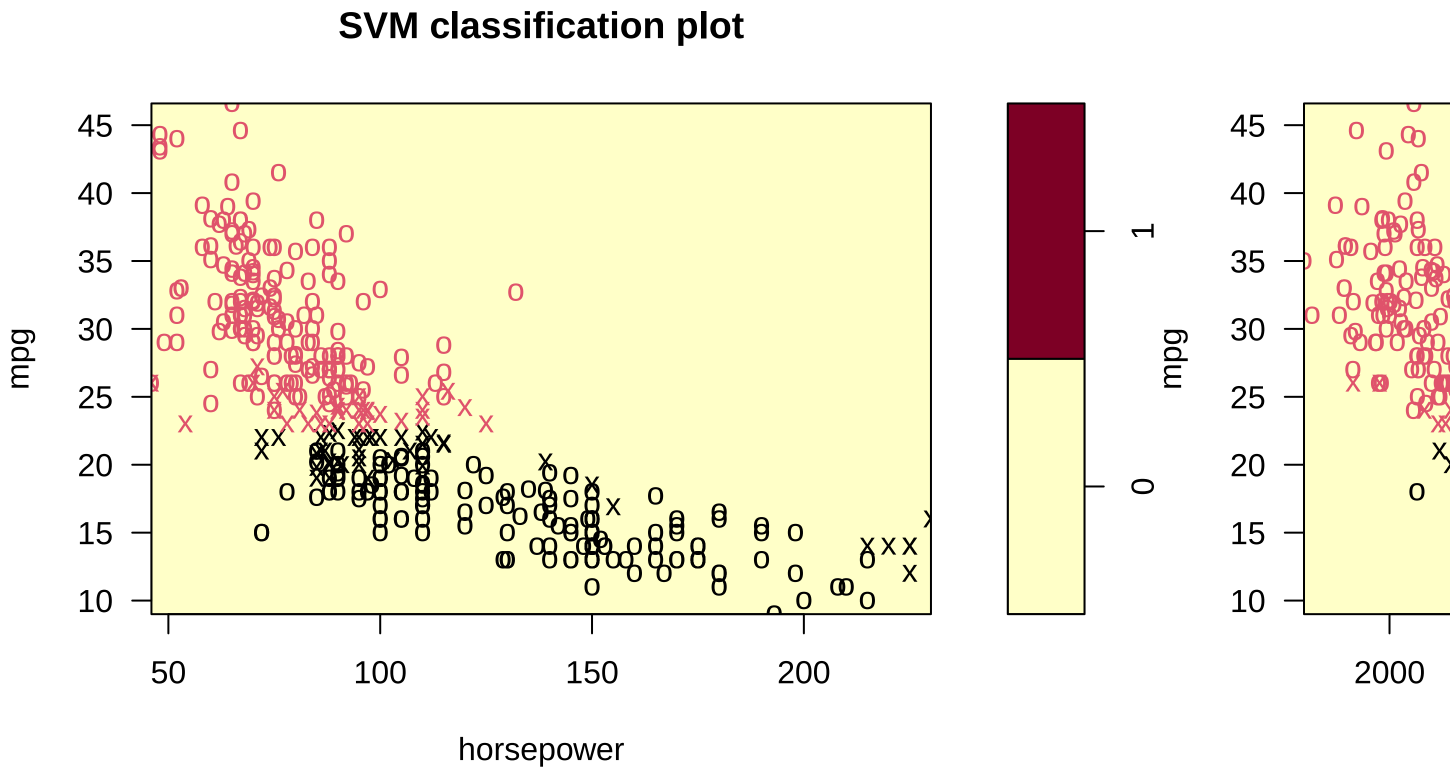

SVM classification plot
mpg
45
40
35
30
25
20
15
10
50
100
150
200
horsepower
1
0
2000

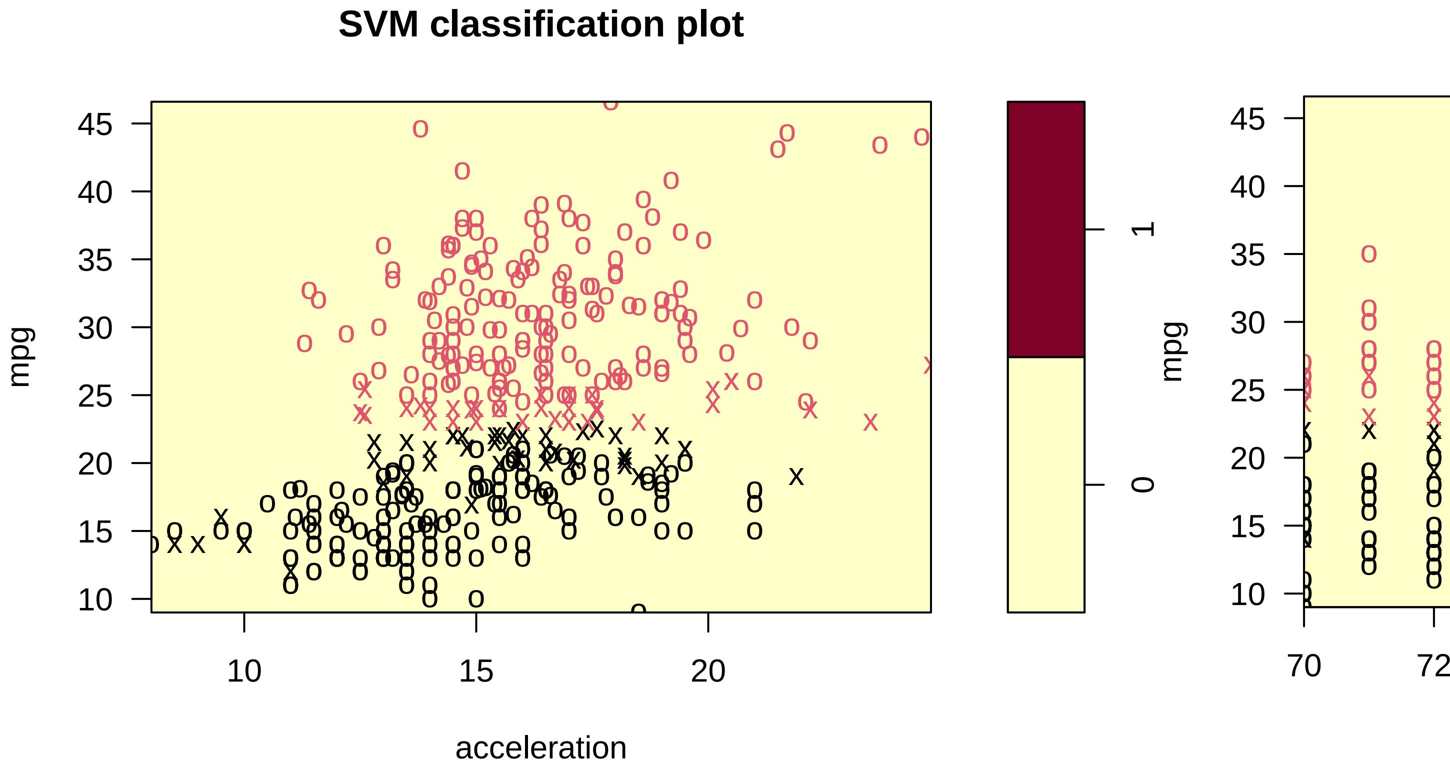

SVM classification plot
mpg
45
40
35
30
25
20
15
10
10
15
20
acceleration
1
0
70
72

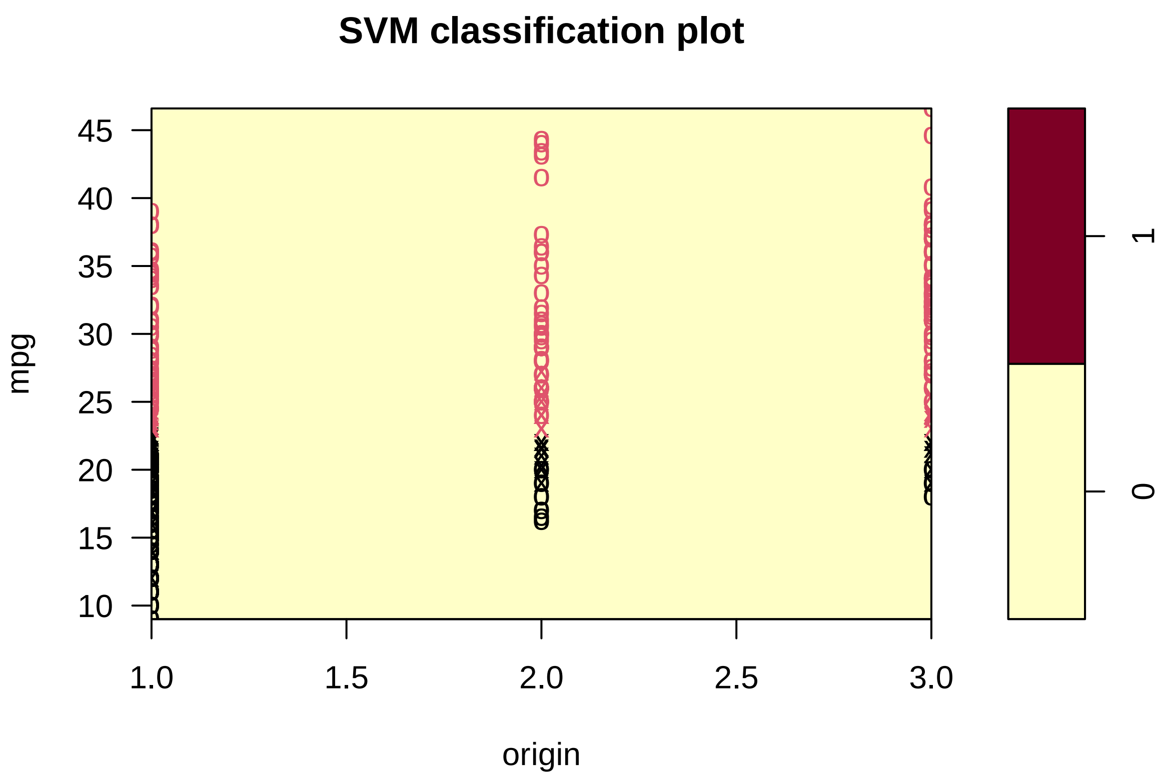



```
svm_plot(fit3)
```




SVM classification plot
mpg
horsepower
50
100
150
200
2000
45
40
35
30
25
20
15
10
1
0


SVM classification plot
mpg
acceleration
10
15
20
70
72
45
40
35
30
25
20
15
10
1
0



Question 9.8

a)

```r
data("OJ")
set.seed(100)
train_oj <- sample(nrow(OJ), 800)
train_OJ <- OJ[train_oj,]
test_OJ <- OJ[-train_oj,]
```

b)

```r
svc_OJ <- svm(Purchase ~ ., data = train_OJ, kernel = "linear", cost = 0.01)
summary(svc_OJ)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train_OJ, kernel = "linear", cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  432
##
##  ( 216 216 )
##
```

```
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

432/800 support vectors were created

c)

```r
set.seed(100)

pred_train_OJ <- predict(svc_OJ, train_OJ)
table(pred_train_OJ, train_OJ$Purchase)
```

```
##
## pred_train_OJ  CH  MM
##            CH 433  78
##            MM  55 234
```

```r
pred_test_OJ <- predict(svc_OJ, test_OJ)
table(pred_test_OJ, test_OJ$Purchase)
```

```
##
## pred_test_OJ  CH  MM
##           CH 147  26
##           MM  18  79
```

Training error rate = 55+78/433+78+55+234 = 16.63% Test error rate = 26+18/147+26+18+79 = 16.30%

d)

```r
tune_OJ <- tune(svm, Purchase ~ ., data = train_OJ, kernel = "linear", ranges = data.frame(cost = seq(0
summary(tune_OJ)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      cost
##  6.555172
##
## - best performance: 0.17
##
## - Detailed performance results:
##          cost   error dispersion
## 1   0.0100000 0.17500 0.04639804
## 2   0.3544828 0.17250 0.03944053
## 3   0.6989655 0.17375 0.04101575
## 4   1.0434483 0.17500 0.03908680
## 5   1.3879310 0.17250 0.03525699
## 6   1.7324138 0.17125 0.03230175
## 7   2.0768966 0.17125 0.03230175
```

```
## 8    2.4213793 0.17125 0.03230175
## 9    2.7658621 0.17125 0.03438447
## 10   3.1103448 0.17375 0.03251602
## 11   3.4548276 0.17250 0.03476109
## 12   3.7993103 0.17250 0.03476109
## 13   4.1437931 0.17250 0.03476109
## 14   4.4882759 0.17125 0.03729108
## 15   4.8327586 0.17125 0.03729108
## 16   5.1772414 0.17125 0.03729108
## 17   5.5217241 0.17125 0.03729108
## 18   5.8662069 0.17125 0.03729108
## 19   6.2106897 0.17125 0.03729108
## 20   6.5551724 0.17000 0.03782269
## 21   6.8996552 0.17000 0.03782269
## 22   7.2441379 0.17000 0.03782269
## 23   7.5886207 0.17000 0.03782269
## 24   7.9331034 0.17000 0.03782269
## 25   8.2775862 0.17000 0.03782269
## 26   8.6220690 0.17000 0.03782269
## 27   8.9665517 0.17000 0.03782269
## 28   9.3110345 0.17000 0.03782269
## 29   9.6555172 0.17000 0.03782269
## 30  10.0000000 0.17000 0.03782269
```

0.7 cost seems to lower the error the most.

e)

```
set.seed(100)

svm_OJ <- svm(Purchase ~ ., data = train_OJ, kernel = "linear", cost = tune_OJ$best.parameters$cost)

svm_train <- predict(svm_OJ, train_OJ)
table(svm_train, train_OJ$Purchase)
```

```
##
## svm_train  CH  MM
##        CH 426  62
##        MM  62 250
```

```
svm_test <- predict(svm_OJ, test_OJ)
table(svm_test, test_OJ$Purchase)
```

```
##
## svm_test  CH  MM
##       CH 143  26
##       MM  22  79
```

Training error rate = 66+61/427+66+61+246 = 15.88% Test error rate = 27+22/143+27+22+78 = 18.15%.

f)

```
set.seed(100)

svm_radial_OJ <- svm(Purchase ~ ., data = train_OJ, kernel = "radial")
```

```r
summary(svm_radial_OJ)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train_OJ, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  368
##
##  ( 187 181 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
svm_radial_train <- predict(svm_radial_OJ, train_OJ)
table(svm_radial_train, train_OJ$Purchase)
```

```
##
## svm_radial_train  CH  MM
##               CH 448  69
##               MM  40 243
```

```r
svm_radial_test <- predict(svm_radial_OJ, test_OJ)
table(svm_radial_test, test_OJ$Purchase)
```

```
##
## svm_radial_test  CH  MM
##              CH 147  32
##              MM  18  73
```

Training error rate = 69+40/448+69+40+243 = 13.63% Test error rate = 32+18/147+32+18+73 = 18.52%

```r
# Tuning - Find cost

svm_radial_tune_OJ <- tune(svm, Purchase ~ ., data = train_OJ, kernel = "radial", ranges = data.frame(c
summary(svm_radial_tune_OJ)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      cost
##  1.387931
##
## - best performance: 0.1575
##
```

```
## - Detailed performance results:
##          cost   error dispersion
## 1   0.0100000 0.39000 0.03809710
## 2   0.3544828 0.17000 0.03545341
## 3   0.6989655 0.16125 0.03747684
## 4   1.0434483 0.15875 0.02949223
## 5   1.3879310 0.15750 0.03016160
## 6   1.7324138 0.15875 0.03175973
## 7   2.0768966 0.16375 0.03793727
## 8   2.4213793 0.16250 0.03632416
## 9   2.7658621 0.16875 0.03784563
## 10  3.1103448 0.17125 0.03729108
## 11  3.4548276 0.17125 0.03634805
## 12  3.7993103 0.17250 0.03476109
## 13  4.1437931 0.17125 0.03488573
## 14  4.4882759 0.16875 0.03346329
## 15  4.8327586 0.16875 0.03346329
## 16  5.1772414 0.16875 0.03346329
## 17  5.5217241 0.17000 0.03446012
## 18  5.8662069 0.17125 0.03120831
## 19  6.2106897 0.17125 0.03120831
## 20  6.5551724 0.17125 0.03120831
## 21  6.8996552 0.17250 0.02993047
## 22  7.2441379 0.17250 0.02993047
## 23  7.5886207 0.17250 0.02993047
## 24  7.9331034 0.17250 0.02993047
## 25  8.2775862 0.17250 0.03162278
## 26  8.6220690 0.17250 0.02813657
## 27  8.9665517 0.17375 0.02664713
## 28  9.3110345 0.17500 0.02700309
## 29  9.6555172 0.17500 0.02700309
## 30 10.0000000 0.17625 0.02791978
```

The optimal cost is around 1

```r
# Tuning - Find errors

set.seed(100)

svm_radial_OJ <- svm(Purchase ~ ., data = train_OJ, kernel = "radial", cost = svm_radial_tune_OJ$best.pa

svm_radial_train <- predict(svm_radial_OJ, train_OJ)
table(svm_radial_train, train_OJ$Purchase)
```

```
##
## svm_radial_train  CH  MM
##               CH 448  70
##               MM  40 242
```

```r
svm_radial_test <- predict(svm_radial_OJ, test_OJ)
table(svm_radial_test, test_OJ$Purchase)
```

```
##
## svm_radial_test  CH  MM
##              CH 147  32
##              MM  18  73
```

Training error rate = 71+43/445+71+43+241 = 14.25% Test error rate = 32+18/147+32+18+73 = 18.52%

g)

```r
poly_OJ <- svm(Purchase ~ ., data = train_OJ, kernel = "polynomial", degree = 2)
summary(poly_OJ)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train_OJ, kernel = "polynomial",
##     degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  447
##
##  ( 228 219 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

447/800 support vectors created.

```r
set.seed(100)

svm_poly_train <- predict(poly_OJ, train_OJ)
table(svm_poly_train, train_OJ$Purchase)
```

```
##
## svm_poly_train  CH  MM
##             CH 458 105
##             MM  30 207
```

```r
svm_poly_test <- predict(poly_OJ, test_OJ)
table(svm_poly_test, test_OJ$Purchase)
```

```
##
## svm_poly_test  CH  MM
##            CH 152  45
##            MM  13  60
```

Training error rate = 105+30/458+105+30+207 = 16.88% Test error rate = 45+13/152+45+13+60 = 21.48%

```r
# Tuning - Find cost
svm_poly_tune <- tune(svm, Purchase ~ ., data = train_OJ, kernel = "polynomial", degree = 2, ranges =
                  data.frame(cost = seq(0.01, 10, length.out = 30)))
summary(svm_poly_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      cost
##  7.588621
##
## - best performance: 0.17125
##
## - Detailed performance results:
##          cost   error dispersion
## 1   0.0100000 0.38875 0.03972562
## 2   0.3544828 0.20250 0.03216710
## 3   0.6989655 0.19750 0.03270236
## 4   1.0434483 0.19250 0.03073181
## 5   1.3879310 0.18250 0.03395258
## 6   1.7324138 0.18375 0.03438447
## 7   2.0768966 0.18125 0.03547789
## 8   2.4213793 0.18000 0.03872983
## 9   2.7658621 0.18125 0.04135299
## 10  3.1103448 0.18250 0.04090979
## 11  3.4548276 0.18250 0.04257347
## 12  3.7993103 0.18000 0.04133199
## 13  4.1437931 0.18125 0.04218428
## 14  4.4882759 0.17875 0.03866254
## 15  4.8327586 0.18000 0.03917553
## 16  5.1772414 0.18000 0.03917553
## 17  5.5217241 0.17875 0.03634805
## 18  5.8662069 0.17875 0.03634805
## 19  6.2106897 0.17875 0.03634805
## 20  6.5551724 0.17625 0.03458584
## 21  6.8996552 0.17500 0.03435921
## 22  7.2441379 0.17375 0.03508422
## 23  7.5886207 0.17125 0.03775377
## 24  7.9331034 0.17250 0.03855011
## 25  8.2775862 0.17250 0.04158325
## 26  8.6220690 0.17250 0.03899786
## 27  8.9665517 0.17500 0.03773077
## 28  9.3110345 0.17750 0.04031129
## 29  9.6555172 0.17750 0.04031129
## 30 10.0000000 0.17750 0.04031129
```

The optimal cost is around 7.6

```r
# Tuning - Find errors

set.seed(100)

svm_poly_tune <- svm(Purchase ~ ., data = train_OJ, kernel = "polynomial", cost = svm_poly_tune$best.pa

poly_train <- predict(svm_poly_tune, train_OJ)
table(poly_train, train_OJ$Purchase)
```

```
##
## poly_train  CH  MM
##         CH 452  72
##         MM  36 240
```

```
poly_test <- predict(svm_poly_tune, test_OJ)
table(poly_test, test_OJ$Purchase)
```

```
##
## poly_test  CH  MM
##        CH 145  35
##        MM  20  70
```

Training error rate = 72+36/452+72+36+240 = 13.50% Test error rate = 35+20/145+35+20+70 = 20.37%

h)

linear kernel SVM 0.7 cost gives the lowest error rate which is 18.15%